Curso Inteligência Artificial: do Zero ao Infinito

Introducão ao Multi-Layer Perceptron

Universidade Federal de Mato Grosso

Agenda

1. Introdução
2. Multi-Layer Perceptron
3. Feed Foward
4. Back Propagation
5. Descending Gradient
6. Multi-Class Problem

Agenda

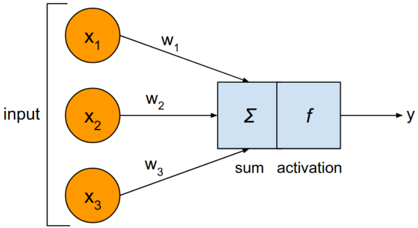
1. Introdução
2. Multi-Layer Perceptron
3. Feed Foward
4. Back Propagation
5. Descending Gradient
6. Multi-Class Problem

Na aula anterior:

- Modelo Perceptron.
- Neurônio Artificial (Sigmoid)

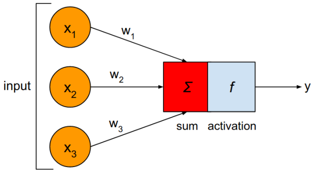
Revisão

Neurônio Artificial



Revisão

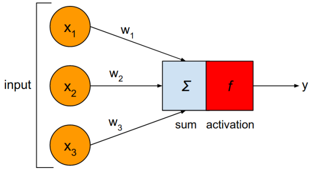
Neurônio Artificial



$$h = \sum_i x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3$$

Revisão

Neurônio Artificial

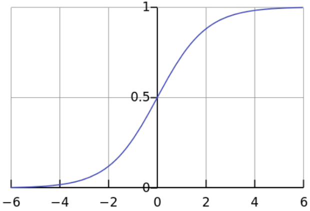


$$h = \sum_i x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3$$

$$\boxed{y = f(h) = f(x_1 w_1 + x_2 w_2 + x_3 w_3)}$$

Revisão

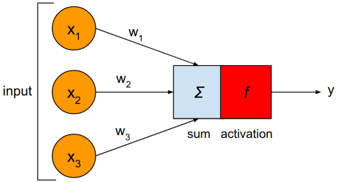
Neurônio Artificial



$$y = \frac{1}{1 + e^{-x}}$$

Revisão

Neurônio Artificial

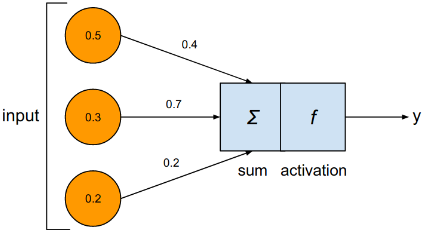


$$h = \sum_i x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3$$

$$y = f(h) = f(x_1 w_1 + x_2 w_2 + x_3 w_3)$$

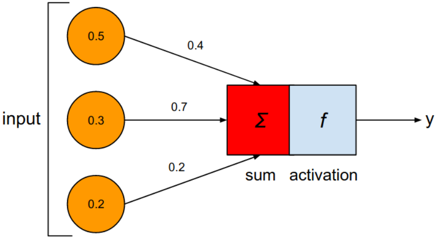
$$\boxed{y = \frac{1}{1 + e^{-(x_1 w_1 + x_2 w_2 + x_3 w_3)}}}$$

Neurônio Artificial
Exemplo



Neurônio Artificial

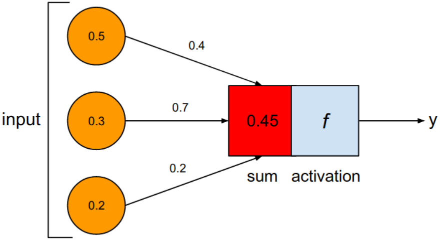
Exemplo



$$\boxed{h = x_1 w_1 + x_2 w_2 + x_3 w_3}$$

Neurônio Artificial

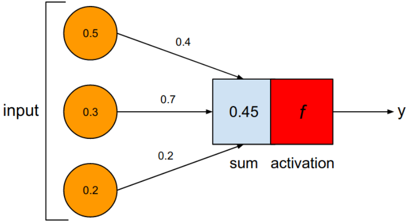
Exemplo



$$h = x_1 w_1 + x_2 w_2 + x_3 w_3 \boxed{= 0.5 \cdot 0.4 + 0.3 \cdot 0.7 + 0.2 \cdot 0.2 = 0.45}$$

Neurônio Artificial

Exemplo



$$h = x_1w_1 + x_2w_2 + x_3w_3 = 0.5 \cdot 0.4 + 0.3 \cdot 0.7 + 0.2 \cdot 0.2 = 0.45$$

$$\boxed{y = \frac{1}{1 + e^{-0.45}} = 0.61}$$

Neurônio Artificial

Resumo

- Neurônios Artificiais são *inspirados* nos neurônios biológicos.
- Transformam um entrada em uma saída usando uma *função de ativação*.
- É uma unidade computacional própria para resolver problemas simples.

Exemplo Prático com Keras

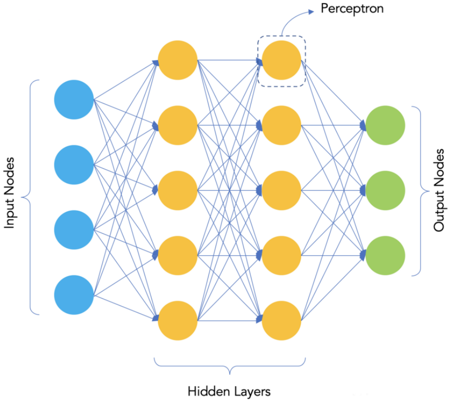
Agenda

1. Introdução
2. Multi-Layer Perceptron
3. Feed Foward
4. Back Propagation
5. Descending Gradient
6. Multi-Class Problem

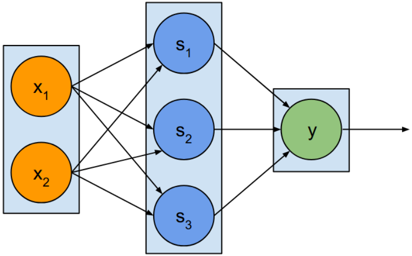
Multi-Layer Perceptron

- O **Multi-Layer Perceptron** é formado por camadas compostas de neurônios artificiais.
- Pode trabalhar com problemas mais complexos.
- O treinamento é feito utilizando o algoritmo *Back Propagation*.

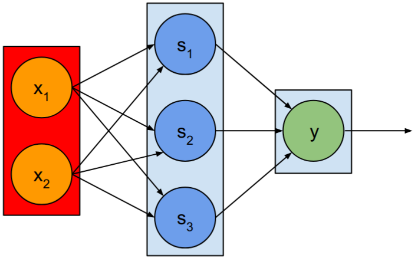
Multi-Layer Perceptron



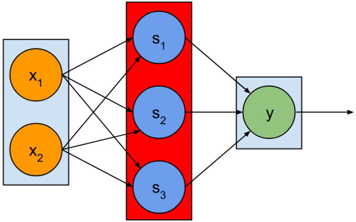
Multi-Layer Perceptron



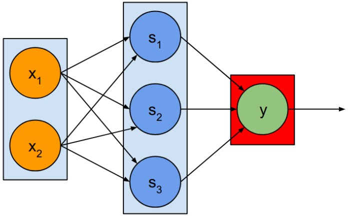
Multi-Layer Perceptron



Multi-Layer Perceptron



Multi-Layer Perceptron



Multi-Layer Perceptron

O processo de treinamento envolve três etapas:

- Feed Forward
- Back Propagation
- Descending Gradient

Multi-Layer Perceptron

Envolve três cálculos:

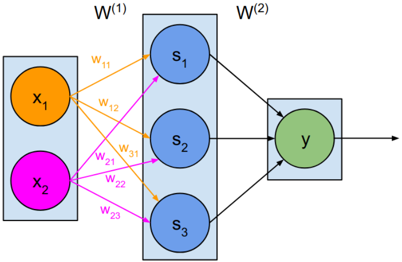
- *Weights* (Pesos a serem calculados)
- *Net Input* (Soma das entradas ponderadas)
- *Activation* (Saída da função de ativação)

Agenda

1. Introdução
2. Multi-Layer Perceptron
3. Feed Foward
4. Back Propagation
5. Descending Gradient
6. Multi-Class Problem

Multi-Layer Perceptron
Feed Forward

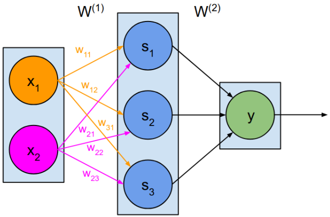
Weights



Multi-Layer Perceptron

Feed Forward

Weights

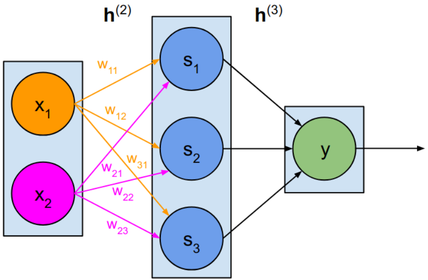


$$W^{(1)} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

Multi-Layer Perceptron

Feed Forward

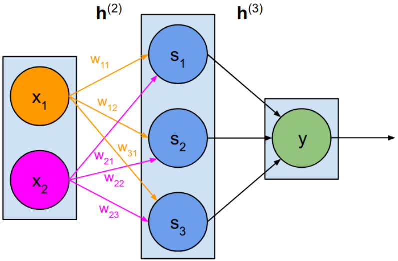
Net Input



Multi-Layer Perceptron

Feed Forward

Net Input

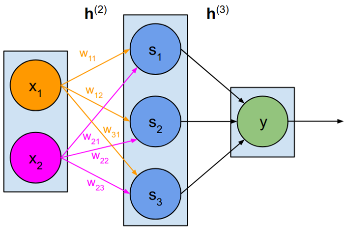


$$\boxed{\mathbf{h}^{(2)} = \mathbf{x}W^{(1)}}$$

Multi-Layer Perceptron

Feed Forward

Net Input

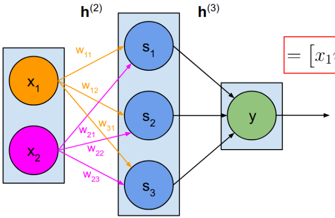


$$\mathbf{h}^{(2)} = \mathbf{x}W^{(1)} = \boxed{\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}}$$

Multi-Layer Perceptron

Feed Forward

Net Input



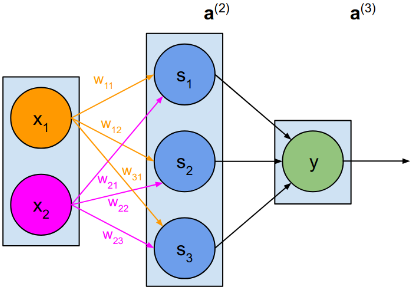
$$\mathbf{h}^{(2)} = \mathbf{x}W^{(1)} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

$$\boxed{= \begin{bmatrix} x_1 w_{11} + x_2 w_{21} & x_1 w_{12} + x_2 w_{22} & x_1 w_{13} + x_2 w_{23} \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix}}$$

Multi-Layer Perceptron

Feed Forward

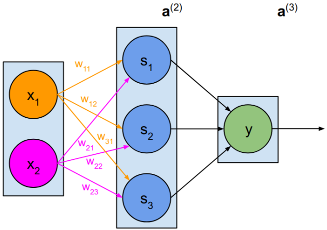
Activation



Multi-Layer Perceptron

Feed Forward

Activation

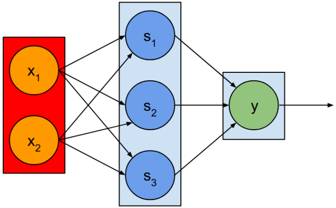


$$\boxed{\mathbf{a}^{(2)} = f(\mathbf{h}^{(2)})}$$

Multi-Layer Perceptron

Feed Forward

1ª Camada

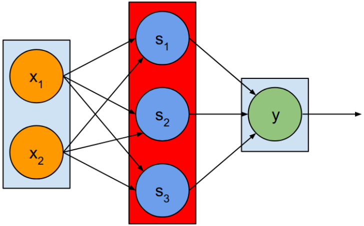


$\mathbf{X}$

Multi-Layer Perceptron

Feed Forward

2ª Camada

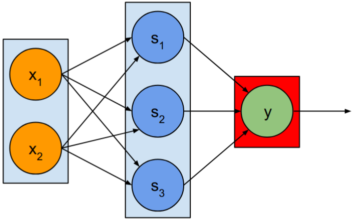


$$\mathbf{h}^{(2)} = \mathbf{x}W^{(1)}$$
$$\mathbf{a}^{(2)} = f(\mathbf{h}^{(2)})$$

Multi-Layer Perceptron

Feed Forward

3ª Camada

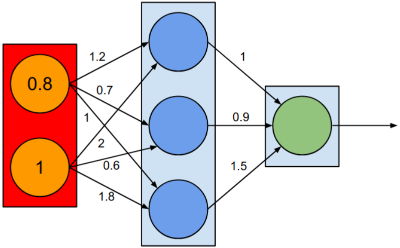


$$\mathbf{h}^{(3)} = \mathbf{a}^{(\mathbf{2})} W^{(2)}$$
$$y = f(\mathbf{h}^{(3)})$$

Exemplo

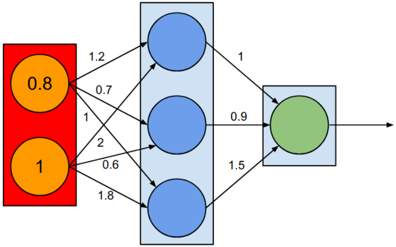
Multi-Layer Perceptron

Exemplo



Multi-Layer Perceptron

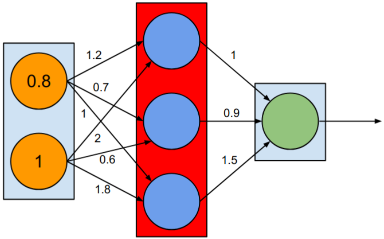
Exemplo



$$\mathbf{x} = \begin{bmatrix} 0.8 & 1 \end{bmatrix}$$

Multi-Layer Perceptron

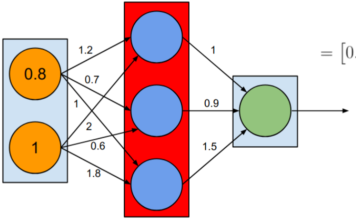
Exemplo



$$\mathbf{h}^{(2)} = \mathbf{x}W^{(1)}$$

Multi-Layer Perceptron

Exemplo

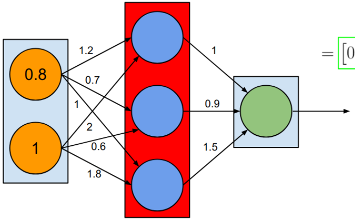


$$\mathbf{h}^{(2)} = \mathbf{x}W^{(1)}$$

$$= \begin{bmatrix} 0.8 & 1 \end{bmatrix} \begin{bmatrix} 1.2 & 0.7 & 1 \\ 2 & 0.6 & 1.8 \end{bmatrix} = \begin{bmatrix} 0.8 \cdot 1.2 + 1 \cdot 2 & 0.8 \cdot 0.7 + 1 \cdot 0.6 & 0.8 \cdot 1 + 1 \cdot 1.8 \end{bmatrix}$$

Multi-Layer Perceptron

Exemplo

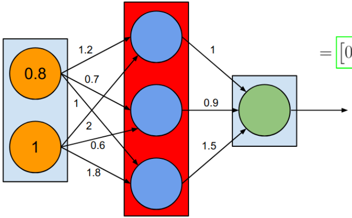


$$\mathbf{h}^{(2)} = \mathbf{x}W^{(1)}$$

$$= \boxed{\begin{bmatrix} 0.8 & 1 \end{bmatrix}} \begin{bmatrix} 1.2 & 0.7 & 1 \\ 2 & 0.6 & 1.8 \end{bmatrix} = \begin{bmatrix} \boxed{0.8\cdot 1.2 + 1\cdot 2} & 0.8\cdot 0.7 + 1\cdot 0.6 & 0.8\cdot 1 + 1\cdot 1.8 \end{bmatrix}$$

Multi-Layer Perceptron

Exemplo

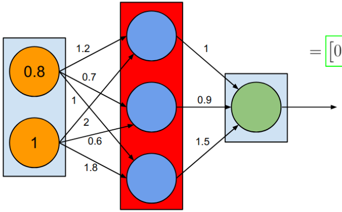


$$\mathbf{h}^{(2)} = \mathbf{x}W^{(1)}$$

$$= \boxed{\begin{bmatrix} 0.8 & 1 \end{bmatrix}} \begin{bmatrix} 1.2 & \boxed{0.7} & 1 \\ 2 & \boxed{0.6} & 1.8 \end{bmatrix} = \begin{bmatrix} 0.8 \cdot 1.2 + 1 \cdot 2 & \boxed{0.8 \cdot 0.7 + 1 \cdot 0.6} & 0.8 \cdot 1 + 1 \cdot 1.8 \end{bmatrix}$$

Multi-Layer Perceptron

Exemplo

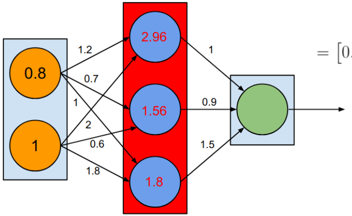


$$\mathbf{h}^{(2)} = \mathbf{x}W^{(1)}$$

$$= \boxed{\begin{bmatrix}0.8 & 1\end{bmatrix}} \begin{bmatrix}1.2 & 0.7 & 1\\ 2 & 0.6 & 1.8\end{bmatrix} = \begin{bmatrix}0.8{\cdot}1.2 + 1{\cdot}2 & 0.8{\cdot}0.7 + 1{\cdot}0.6 & \boxed{0.8{\cdot}1 + 1{\cdot}1.8}\end{bmatrix}$$

Multi-Layer Perceptron

Exemplo



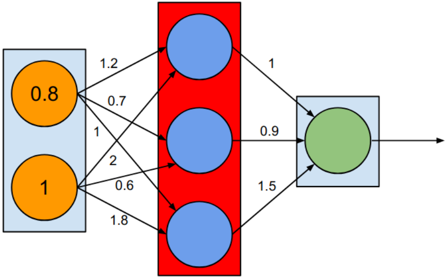
$$\mathbf{h}^{(2)} = \mathbf{x}W^{(1)}$$

$$= \begin{bmatrix} 0.8 & 1 \end{bmatrix} \begin{bmatrix} 1.2 & 0.7 & 1 \\ 2 & 0.6 & 1.8 \end{bmatrix} = \begin{bmatrix} 0.8 \cdot 1.2 + 1 \cdot 2 & 0.8 \cdot 0.7 + 1 \cdot 0.6 & 0.8 \cdot 1 + 1 \cdot 1.8 \end{bmatrix}$$

$$\boxed{= \begin{bmatrix} 2.96 & 1.56 & 1.8 \end{bmatrix}}$$

Multi-Layer Perceptron

Exemplo

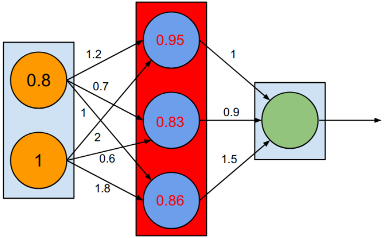


$$\mathbf{h}^{(2)} = \begin{bmatrix} 2.96 & 1.56 & 1.8 \end{bmatrix}$$

$$\mathbf{a}^{(2)} = \frac{1}{1 + e^{-\mathbf{h}^{(2)}}}$$

Multi-Layer Perceptron

Exemplo

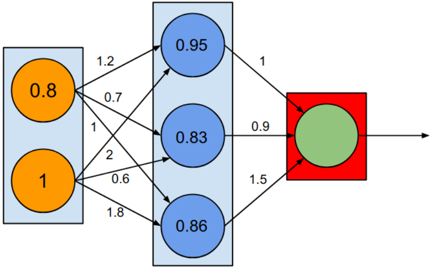


$$\mathbf{h}^{(2)} = \begin{bmatrix} 2.96 & 1.56 & 1.8 \end{bmatrix}$$

$$\mathbf{a}^{(2)} = \frac{1}{1 + e^{-\mathbf{h}^{(2)}}} = \boxed{\begin{bmatrix} 0.95 & 0.83 & 0.86 \end{bmatrix}}$$

Multi-Layer Perceptron

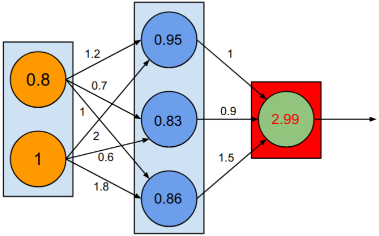
Exemplo



$$\mathbf{h}^{(3)} = \mathbf{a}^{(2)} W^{(2)}$$

Multi-Layer Perceptron

Exemplo

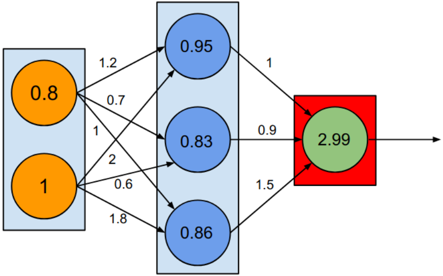


$$\mathbf{h}^{(3)} = \mathbf{a}^{(2)} W^{(2)}$$

$$\boxed{= \begin{bmatrix} 0.95 & 0.83 & 0.86 \end{bmatrix} \begin{bmatrix} 1 \\ 0.9 \\ 1.5 \end{bmatrix} = 2.99}$$

Multi-Layer Perceptron

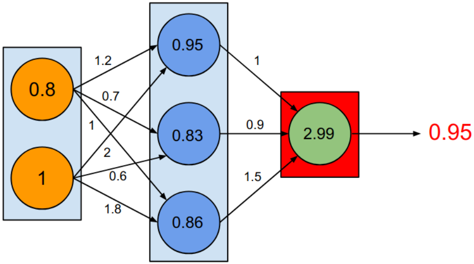
Exemplo



$$y = \frac{1}{1 + e^{-\mathbf{h}^{(3)}}}$$

Multi-Layer Perceptron

Exemplo



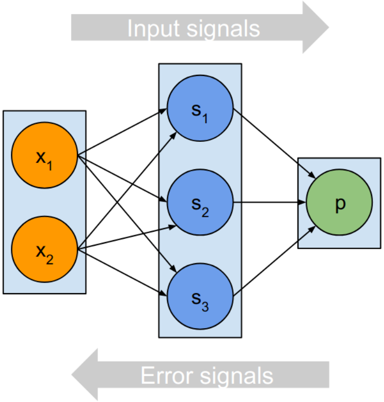
$$y = \frac{1}{1 + e^{-\mathbf{h}^{(3)}}}$$

$$\boxed{= \frac{1}{1 + e^{-2.99}} = 0.95}$$

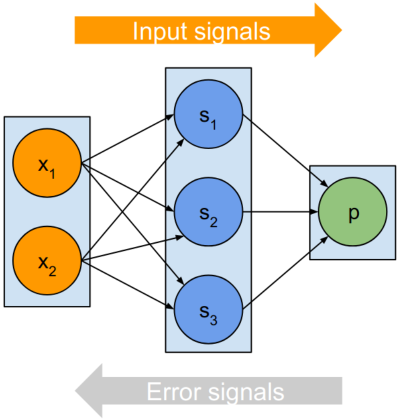
Agenda

1. Introdução
2. Multi-Layer Perceptron
3. Feed Foward
4. Back Propagation
5. Descending Gradient
6. Multi-Class Problem

Processo de Treinamento

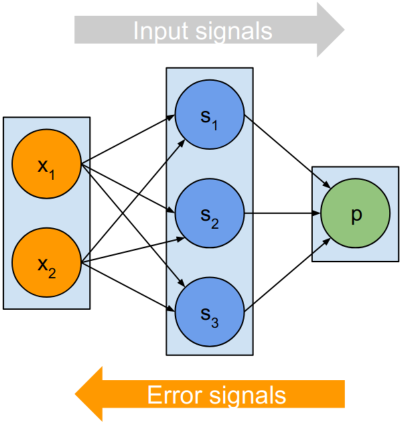


Processo de Treinamento



1. Obtém uma predição
   - *feed forward*
2. Calcula o erro
   - *função loss*

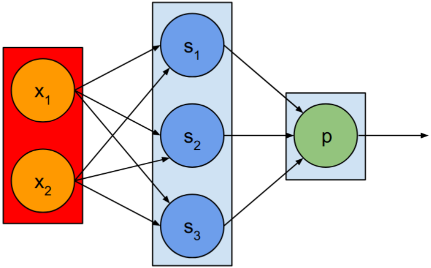
Processo de Treinamento



3. Calcula o gradiente
   - *back propagation*
4. Atualiza os pesos
   - *gradiente descendente*

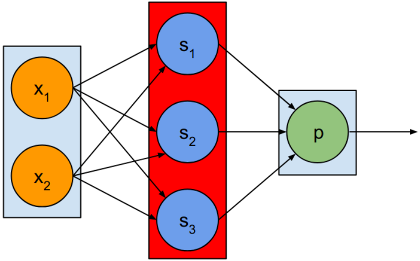
Processo de Treinamento

Obter Predição

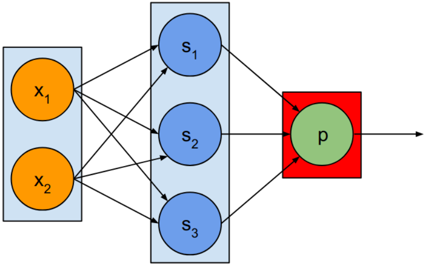


Processo de Treinamento

Obter Predição

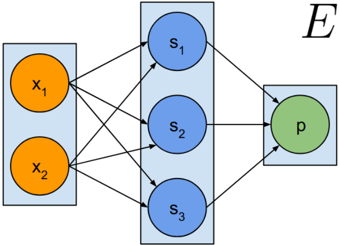


Processo de Treinamento
Obter Predição



Processo de Treinamento

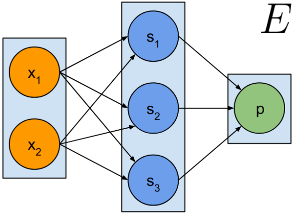
Calcula o erro



$$E = E(\mathbf{p}, \mathbf{y})$$

Processo de Treinamento

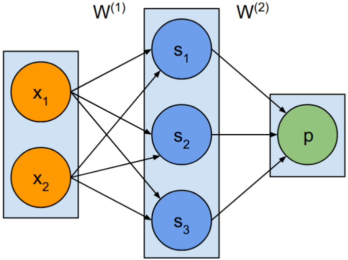
Calcula o erro



$$E = E(\mathbf{p}, \mathbf{y}) = \boxed{\frac{1}{2}(\mathbf{p} - \mathbf{y})^2}$$

Processo de Treinamento

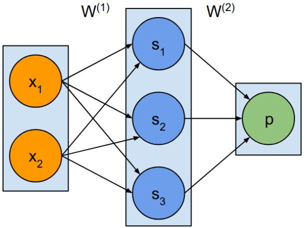
Back Propagation



$$F = F(\mathbf{x}, W)$$

Processo de Treinamento

Back Propagation

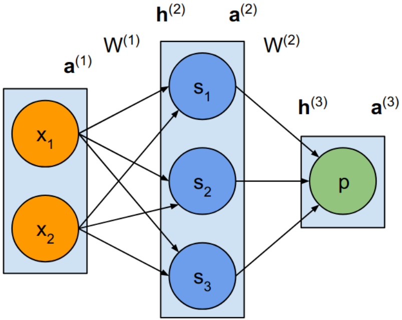


$$F = F(\mathbf{x}, W)$$

$$E = E(\mathbf{p}, \mathbf{y}) = E(F(\mathbf{x}, W), \mathbf{y})$$

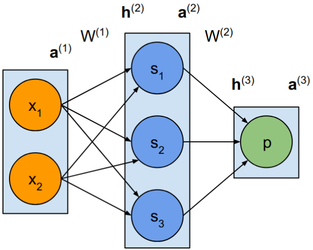
Processo de Treinamento

Back Propagation



Processo de Treinamento

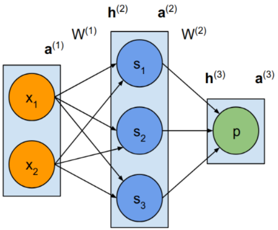
Back Propagation



$$\frac{\partial E}{\partial W^{(2)}} = \frac{\partial E}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial h^{(3)}} \frac{\partial h^{(3)}}{\partial W^{(2)}}$$

Processo de Treinamento

Back Propagation



$$\frac{\partial E}{\partial W^{(2)}} = \boxed{\frac{\partial E}{\partial a^{(3)}}}\boxed{\frac{\partial a^{(3)}}{\partial h^{(3)}}}\boxed{\frac{\partial h^{(3)}}{\partial W^{(2)}}}$$

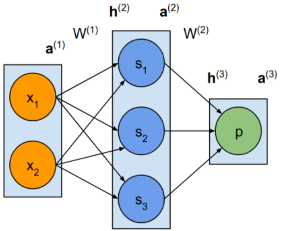
$$\frac{\partial E}{\partial W^{(2)}} = \boxed{(a^{(3)} - y)}\boxed{\sigma'(h^{(3)})}\boxed{a^{(2)}}$$

Processo de Treinamento

Back Propagation

$$\frac{\partial E}{\partial W^{(1)}} = \boxed{\frac{\partial E}{\partial a^{(3)}}}\boxed{\frac{\partial a^{(3)}}{\partial h^{(3)}}}\boxed{\frac{\partial h^{(3)}}{\partial a^{(2)}}}\boxed{\frac{\partial a^{(2)}}{\partial h^{(2)}}}\boxed{\frac{\partial h^{(2)}}{\partial W^{(1)}}}$$

$$\frac{\partial E}{\partial W^{(1)}} = \boxed{(a^{(3)} - y)}\boxed{\sigma'(h^{(3)})}\boxed{W^{(2)}}\boxed{\sigma'(h^{(2)})}\boxed{x}$$



Processo de Treinamento

Back Propagation

$$\frac{\partial E}{\partial W^{(2)}} = (a^{(3)} - y)\sigma'(h^{(3)})a^{(2)}$$

$$\frac{\partial E}{\partial W^{(1)}} = (a^{(3)} - y)\sigma'(h^{(3)})W^{(2)}\sigma'(h^{(2)})x$$

Agenda

1. Introdução
2. Multi-Layer Perceptron
3. Feed Foward
4. Back Propagation
5. Descending Gradient
6. Multi-Class Problem

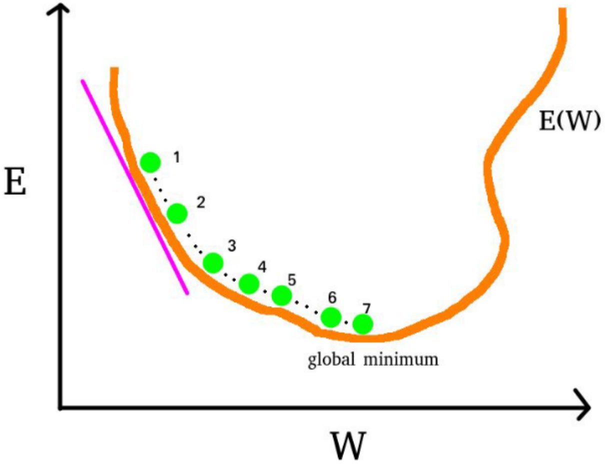
Descending Gradient

- O **Gradiente Descendente** é um algoritmo de otimização usado para treinar redes neurais.
- Sua função é encontrar um conjunto de parâmetros que minimizam uma função custo.
- É usado quando os parâmetros não podem ser calculados analiticamente e devem ser encontrados por um algoritmo de otimização.

Referência: https://machinelearningmastery.com/gradient-descent-for-machine-learning/

Processo de Treinamento

Descending Gradient



Descending Gradient

Pseudo-Código

Algoritmo 1: Gradiente Descendente Estocástico

Entrada: $\vec{x} = \{(x_1, ...x_n)\}$ $\vec{y} = \{y_1, ...y_n\}\}$
Saída: $\vec{w}$

início
  Inicializa $\vec{w}$ e b com valores aleatórios;
  repita
    para *cada amostra* (x_i, y_i) *de* $(\vec{x}, \vec{y})$ **faça**
      Calcule $\hat{y}_i \leftarrow \vec{w}x_i + b$;
      Calcule a Loss: $\mathcal{L}_i \leftarrow L(\hat{y}_i, y_i)$;
      Calcule o gradiente: $\Delta w \leftarrow -\nabla \mathcal{L}_i(w); \Delta b \leftarrow -\frac{\partial \mathcal{L}_i}{\partial b}$;
      `// Atualiza os parâmetros`
      $w \leftarrow w + \alpha \Delta w$; $b \leftarrow b + \alpha \Delta b$;
  até *n épocas*;

Exemplo Prático com Keras

Descending Gradient

Variações

- **Gradiente Descendente Estocástico**
  - A atualização dos coeficientes é realizada para cada instância de treinamento.
  - *Batch-Size* = 1.
  - Na versão verdadeiramente estocástica a amostra extraída é devolvida ao conjunto.
- **Gradiente Descendente em Lote**
  - A atualização dos coeficientes é realizada após o cálculo da *Loss* de todas as instâncias de treinamento.
  - *Batch-Size* igual ao tamanho do conjunto de treinamento.
  - É chamado *Batch Gradient Descent* ou *Vanilla Gradient Descent*.
- **Gradiente Descendente em Mini-Lote**
  - A atualização dos coeficientes é realizada após o cálculo da *Loss* de um conjunto de instâncias de treinamento.
  - Quanto maior o tamanho do *Batch-Size* mais estável é o treinamento.

Referência: https://sebastianraschka.com/faq/docs/sgd-methods.html

Exemplo Prático: Analisando o impacto do tamanho do batch

Agenda

1 Introdução

2 Multi-Layer Perceptron

3 Feed Foward

4 Back Propagation

5 Descending Gradient

6 Multi-Class Problem

Multi-Class Problem

- Em problemas de classificação binária utiliza-se a função *Sigmoid* como função de ativação
- A função Sigmoide produz como saída um valor entre zero e um.
- Como função Loss, utiliza-se a *Binary Cross Entropy*.

Multi-Class Problem

- Para problemas **multiclasse**, utiliza-se a função *Softmax* como função de ativação.
- A função Softmax estende o conceito da função Sigmoid para problemas multiclasse.
- Softmax atribui uma probabilidade (entre 0 e 1) para cada classe sendo que o somatório é igual a 1.

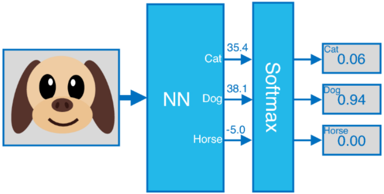
Referência: Multi-Class Neural Networks: Softmax

Multi-Class Problem
Softmax

Função de Ativação **Softmax**:

- Utilizada para **classificação multiclasse**
- Pode-se interpretar sua saída como a **confiança** de uma amostra pertencer a uma classe.
    - Confiança $\neq$ Probabilidade
- Saída deve estar no formato **one-hot enconding**
    - a saída de um neurônio depende dos outros neurônios de saída.

$$y_i(x) = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_i}}$$



$$y \in [0, 1], \sum y_i = 1$$

Funções de Ativação

One-Hot Encoding

y	one-hot				
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	0	1

y	one-hot				
3	0	0	0	1	0
4	0	0	0	0	1
1	0	1	0	0	0
0	1	0	0	0	0
3	0	0	0	1	0

Multi-Class Problem

Categorical Cross Entropy

- A função Loss *Categorical Cross Entropy* é adequada para problemas de classificação multi-classe.
- Seu cálculo é dado pela seguinte fórmula:

$$J(z) = \frac{-1}{N} \sum_{j}^{M} \sum_{i}^{N} y_{ij} \log(\hat{y}_{ij}) + (1 - y_{ij}) \log(1 - \hat{y}_{ij})$$

Exemplo Prático com Keras e Iris dataset

Learning Rate

Exemplo Keras

- A **taxa de aprendizagem** é um hiperparâmetro configurável usado no treinamento de redes neurais que possui um pequeno valor (entre 0,0 e 1,0).
- Valores menores requerem mais *épocas* de treinamento devido às mudanças menores feitas nos pesos a cada atualização.
- Valores maiores resultam em mudanças rápidas e requerem menos períodos de treinamento.

Learning Rate

Exemplo Keras

- O desafio de treinar redes neurais de aprendizado profundo envolve a seleção cuidadosa da taxa de aprendizado.
- Uma taxa de aprendizado muito grande pode fazer com que o modelo convirja muito rapidamente para uma solução abaixo do ideal,
- Um valor muito pequeno pode fazer com que o processo de aprendizado pare.

Referência: https://www.deeplearningbook.com.br/o-efeito-da-taxa-de-aprendizagem-no-treinamento-de-redes-neurais-artificiais/

Exemplo Prático: Analisando o impacto do valor da taxa de aprendizagem

Redes Neurais Multicamadas

Referências

- Neural networks: training with backpropagation.
  - https://www.jeremyjordan.me/neural-networks-training/
- A Step by Step Backpropagation Example
  - https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
- Deep Learning Book: Capítulo 37 - O Efeito do Batch Size no Treinamento de Redes Neurais Artificiais
  - https://www.deeplearningbook.com.br/o-efeito-do-batch-size-no-treinamento-de-redes-neurais-artificiais/
- Deep Learning Book: Capítulo 38 - O Efeito da Taxa de Aprendizagem no Treinamento de Redes Neurais Artificiais
  - https://www.deeplearningbook.com.br/o-efeito-da-taxa-de-aprendizagem-no-treinamento-de-redes-neurais-artificiais/
- Google Tensorflow Playground
  - http://playground.tensorflow.org/

Curso Inteligência Artificial: do Zero ao Infinito

Introducão ao Multi-Layer Perceptron

Universidade Federal de Mato Grosso